Data Quality: From Theory to Practice

Data Quality: From Theory to Practice

Wenfei Fan
School of Informatics, University of Edinburgh, and RCBD, Beihang University
wenfei@inf.ed.ac.uk

ABSTRACT

Data quantity and data quality, like two sides of a coin, are equally important to data management. This paper provides an overview of recent advances in the study of data quality, from theory to practice. We also address challenges introduced by big data to data quality management.

1. INTRODUCTION

When we talk about big data, we typically emphasize the quantity (volume) of the data. We often focus on techniques that allow us to efficiently store, manage and query the data. For example, there has been a host of work on developing scalable algorithms that, given a query Q and a dataset D, compute query answers $Q(D)$ when D is big.

But can we trust $Q(D)$ as correct query answers?

EXAMPLE 1. *In table D_0 of Fig. 1, each tuple specifies the name (*FN, LN*), phone (country code* CC*, area code* AC*,* landline*,* mobile*), address (*street*,* city *and* zip*), and marital* status *of an employee. Consider the following queries.*

(1) Query Q_1 is to find distinct employees in Edinburgh whose first name is Mary. A textbook answer to Q_1 in D_0 is that $Q_1(D_0)$ consists of tuples t_2 and t_3.

However, there are at least three reasons that discredit our trust in $Q_1(D_0)$. (a) In tuple t_1, attribute t_1[AC] is 131, which is the area code of Edinburgh, not of London. Hence t_1 is "inconsistent", and t_1[city] may actually be Edinburgh. (b) Tuples t_2 and t_3 may refer to the same person, i.e., they may not be "distinct"*. (c) Relation D_0 may be incomplete: there are possibly employees in Edinburgh whose records are not included in D_0. In light of these, we do not know whether $Q_1(D_0)$ gives us all correct answers.*

(2) Suppose that t_1, t_2 and t_3 refer to the same Mary, and that they were once correct records (except the address of t_1). Query Q_2 is to find her current last name. It is not clear whether the answer is Smith or Luth. Indeed, some attributes of t_1, t_2 and t_3 have become obsolete and thus inaccurate. □

The example shows that if the quality of the data is bad, we cannot find correct query answers no matter how scalable and efficient our query evaluation algorithms are.

Unfortunately, real-life data is often dirty: inconsistent, inaccurate, incomplete, obsolete and duplicated. Indeed, "more than 25% of critical data in the world's top companies is flawed" [53], and "pieces of information perceived as being needed for clinical decisions were missing from 13.6% to 81% of the time" [76]. It is also estimated that "2% of records in a customer file become obsolete in one month" [31] and hence, in a customer database, 50% of its records may be obsolete and inaccurate within two years.

Dirty data is costly. Statistics shows that "bad data or poor data quality costs US businesses $600 billion annually" [31], "poor data can cost businesses 20%-35% of their operating revenue" [92], and that "poor data across businesses and the government costs the US economy $3.1 trillion a year" [92]. Worse still, when it comes to big data, the scale of the data quality problem is historically unprecedented.

These suggest that quantity and quality are equally important to big data, *i.e., big data = data quantity + data quality*.

This paper aims to provide an overview of recent advances in the study of data quality, from fundamental research (Section 2) to practical techniques (Section 3). It also identifies challenges introduced by big data to data quality management (Section 4). Due to the space constraint, this is by no means a comprehensive survey. We opt for breadth rather than depth in the presentation. Nonetheless, we hope that the paper will incite interest in the study of data quality management for big data. We refer the interested reader to recent surveys on the subject [7, 11, 37, 52, 62, 78].

	FN	LN	CC	AC	landline	mobile	street	city	zip	status
t_1:	Mary	Smith	44	131	3855662	7966899	5 Crichton	London	W1B 1JL	single
t_2:	Mary	Luth	44	131	*null*	*null*	10 King's Road	Edinburgh	EH4 8LE	married
t_3:	Mary	Luth	44	131	6513877	7966899	8 Mayfield	Edinburgh	EH4 8LE	married
t_4:	Bob	Webber	01	908	6512845	3393756	PO Box 212	Murray Hill	NJ 07974	single
t_5:	Robert	Webber	01	908	6512845	*null*	9 Elm St.	Murray Hill	NJ 07974	single

Figure 1: An employee dataset D_0

2. FOUNDATIONS OF DATA QUALITY

Central to data quality are data consistency, data deduplication, information completeness, data currency and data accuracy. The study of data quality has been mostly focusing on data consistency and deduplication in relational data. Nonetheless, each and every of the five central issues introduces fundamental problems. In this section we survey fundamental research on these issues. We highlight dependency-based approaches since they may yield a uniform logical framework to handle these issues.

2.1 Data Consistency

Data consistency refers to the validity and integrity of data representing real-world entities. It aims to detect errors (inconsistencies and conflicts) in the data, typically identified as violations of *data dependencies* (integrity constraints). It is also to help us *repair* the data by fixing the errors.

There are at least two questions associated with data consistency. What data dependencies should we use to detect errors? What repair model do we adopt to fix the errors?

Data dependencies. Several classes of data dependencies have been studied as data quality rules, including

- functional dependencies (FDs) and inclusion dependencies (INDs) [14, 23] found in textbooks (*e.g.,* [1]);
- conditional functional dependencies (CFDs) [38] and conditional inclusion dependencies (CINDs) [15], which extend FDs and INDs, respectively, with a pattern tableau of semantically related constants;
- denial constraints (DCs) [8, 23], which are universally quantified first-order logic (FO) sentences of the form $\forall \bar{x} \neg(\phi(\bar{x}) \wedge \beta(\bar{x}))$, where $\phi(\bar{x})$ is a non-empty conjunction of relation atoms over $\bar{x}$, and $\beta(\bar{x})$ is a conjunction of built-in predicates $=, \neq, <, >, \leq, \geq$;
- equality-generating dependencies [2] (EGDs [9]), a special case of DCs when $\beta(\bar{x})$ is of the form $x_i = x_j$; our familiar FDs are a special case of EGDs;
- tuple-generating dependencies [2] (TGDs [9]), FO sentences of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x}, \bar{y}))$, where $\phi(\bar{x})$ and $\psi(\bar{x}, \bar{y})$ are conjunctions of relation atoms over $\bar{x}$ and $\bar{x} \cup \bar{y}$, respectively, such that each variable of $\bar{x}$ occurs in at least one relation atom of $\phi(\bar{x})$;
- full TGDs [2], special case of TGDs without existential quantifiers, *i.e.,* of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow \psi(\bar{x}))$; and
- LAV TGDs [2], a special case of TGDs in which $\phi(\bar{x})$ is a single relation atom; LAV TGDs subsume INDs.

EXAMPLE 2. *We may use the following* CFDs *as data quality rules on the employee relation of Figure 1:*

$$\varphi_1 = ((\mathsf{CC}, \mathsf{zip} \rightarrow \mathsf{street}), T_{P1}),$$
$$\varphi_2 = ((\mathsf{CC}, \mathsf{AC} \rightarrow \mathsf{city}), T_{P2}),$$

where CC, zip → street *and* CC, AC → city *are* FDs *embedded in the* CFDs*, and T_{P1} and T_{P2} are pattern tableaux:*

T_{P1}:

CC	zip	street
44	_	_

T_{P2}:

CC	AC	city
_	_	_
44	*131*	*Edinburgh*
01	*908*	*Murray Hill*

CFD φ_1 *states that in the UK (when* CC = *44),* zip *code uniquely determines* street*. In other words,* CC, zip → street *is an* FD *that is enforced only on tuples that match the pattern* CC = *44, e.g., on t_1–t_3 in D_0, but not on t_4–t_5. Taking φ as a data quality rule, we find that t_2 and t_3 violate φ_1 and hence, are inconsistent: they have the same* zip *but differ in* street*. Such errors cannot be caught by conventional* FDs*.*

CFD φ_2 *says that country code* CC *and area code* AC *uniquely determine* city*. Moreover, in the UK (i.e.,* CC = *44), when* AC *is 131,* city *must be Edinburgh; and in the US (*CC = *01), if* AC *is 908, then* city *is Murray Hill. It catches t_1 as a violation, i.e., a single tuple may violate a* CFD*. Note that φ_2 subsumes conventional* FD CC, AC → city*, as indicated by the first tuple in T_{P2}, in which '_' is a "wildcard" that matches any value (see [38] for details).* □

To decide what class of dependencies we should use as data quality rules, we want to strike a balance between its "expressive power", *i.e.,* whether it is capable

Dependencies	Implication
FDs	$O(n)$ (cf. [1])
INDs	PSPACE-complete (cf. [1])
FDs + INDs	undecidable (cf. [1])
CFDs	coNP-complete [38]
CINDs	EXPTIME-complete [15]
CFDs + CINDs	undecidable [15]
DCs	coNP-complete [8]
TGDs	undecidable (cf. [1])

Table 1: Complexity of implication analysis

of catching errors commonly found in practice, and the complexity for reasoning about its dependencies and for repairing data.

There are two classical problems for reasoning about dependencies: the satisfiability and implication problems.

Satisfiability. For a class $\mathcal{C}$ of dependencies and $\varphi \in \mathcal{C}$, we use $D \models \varphi$ to denote that a database D satisfies φ, depending on how $\mathcal{C}$ is defined. For a set $\Sigma \subseteq \mathcal{C}$, we use $D \models \Sigma$ to denote that D satisfies all dependencies in Σ. The *satisfiability problem* for $\mathcal{C}$ is to decide, given a finite set $\Sigma \subseteq \mathcal{C}$ defined on a relational schema $\mathcal{R}$, whether there exists a nonempty finite instance D of $\mathcal{R}$ such that $D \models \Sigma$. That is, whether the data quality rules in Σ are consistent themselves.

We can specify arbitrary FDs without worrying about their satisfiability. Indeed, every set of EGDs (or TGDs) can be satisfied by a single-tuple relation [8]. However, a set of DCs or CFDs may *not* be satisfiable by a nonempty database. While the satisfiability problem for DCs has not been settled, it is known that it is NP-complete for CFDs [38], owing to the constant patterns in CFDs. That is, the expressive power of CFDs and DCs come at a price of a higher complexity.

Implication. Consider a finite set $\Sigma \subseteq \mathcal{C}$ of dependencies and another $\varphi \in \mathcal{C}$, both defined on instances of a relational schema $\mathcal{R}$. We say that Σ *implies* φ, denoted by $\Sigma \models \varphi$, if for all instances D of $\mathcal{R}$, $D \models \varphi$ as long as $D \models \Sigma$. The *implication problem* for $\mathcal{C}$ is to decide, given $\Sigma \subseteq \mathcal{C}$ and $\varphi \in \mathcal{C}$ over a relational schema $\mathcal{R}$, whether $\Sigma \models \varphi$. The implication analysis helps us remove redundant data quality rules and hence, speed up error detection and data repairing processes.

Table 1 summarizes known complexity of the implication analysis of data dependencies used as data quality rules.

Data repairing. There are two approaches to obtaining consistent information from an inconsistent database, both proposed by [6]: *data repairing* is to find another database that is consistent and minimally differs from the original database; and *consistent query answering* is to find an answer to a given query in every repair of the original database. Both approaches are based on the notion of repairs. We focus on data repairing in this paper, and refer the interested reader to a comprehensive survey [11] and recent work [12, 67, 86, 87] on consistent query answering.

Repair models. Assume a function $\mathsf{cost}(D, D_r)$ that measures the difference between instances D and D_r of a relational schema $\mathcal{R}$, such that the smaller it is, the closer D_r is to D. Given a set Σ of dependencies and an instance D of $\mathcal{R}$, a *repair* of D *relative to* Σ and $\mathsf{cost}(,)$ is an instance D_r of $\mathcal{R}$ such that $D_r \models \Sigma$ and $\mathsf{cost}(D, D_r)$ is minimum among all instances of $\mathcal{R}$ that satisfy Σ. Several repair models have been studied, based on how $\mathsf{cost}(D, D_r)$ is defined:

- *S-repair* [23]: $\mathsf{cost}(D, D_r) = |D \setminus D_r|$, where $D_r \subseteq D$; assuming that the information in D is inconsistent but complete, this model allows tuple deletions only;
- C-repair [6]: $\mathsf{cost}(D, D_r) = |D \oplus D_r|$, where $D \oplus D_r$ is defined as $(D \setminus D_r) \cup (D_r \setminus D)$; assuming that D is neither consistent nor complete, this model allows both tuple deletions and tuple insertions;
- *CC-repair* [2]: a C-repair such that $|D \oplus D_r|$ is strictly smaller than $|D \oplus D'_r|$ for all D'_r that satisfies Σ; and
- *U-repair* [91, 14]: $\mathsf{cost}(D, D_r)$ is a numerical aggregation function defined in terms of distances and accuracy of attribute values in D and D_r; this model supports attribute value modifications.

For example, the repair model of [14] assumes (a) a *weight* $w(t, A)$ associated with each attribute A of each tuple t in D, and (b) a *distance* function $\mathsf{dis}(v, v')$ for values v and v' in the same domain. Intuitively, $w(t, A)$ indicates the confidence in the *accuracy* of $t[A]$, and $\mathsf{dis}(v, v')$ measures how close v' is to v. The cost of changing the value of an attribute $t[A]$ from v to v' is defined as: $\mathsf{cost}(v, v') = w(t, A) \cdot \mathsf{dis}(v, v')$. That is, the more accurate the original $t[A]$ value v is and the more distant the new value v' is from v, the higher the cost of the change is. The cost of changing a tuple t to t' is the sum of $\mathsf{cost}(t[A], t'[A])$ for A ranging over all attributes in t in which the value of $t[A]$ is modified. The cost of changing D to D_r, denoted by $\mathsf{cost}(D, D_r)$, is the sum of the costs of modifying tuples in D. In practice, repairing is typically carried out via U-repair (see Section 3).

The repair checking problem. Consider a class $\mathcal{C}$ of dependencies and a repair model T with which function $\mathsf{cost}_T(,)$ is associated. The *repair checking problem* for

Dependencies	Repair model	Repair checking
full TGDs	S-repair	PTIME [85]
one FD + one IND	S-repair	coNP-complete [23]
DCs	S-repair	LOGSPACE (cf. [2])
WA LAV TGDs + EGDs	S-repair	LOGSPACE [2]
full TGDs + EGDs	S-repair	PTIME-complete [2]
WA TGDs + EGDs	S-repair	coNP-complete [2]
DCs	C-repair	coNP-complete [73]
full TGDs + EGDs	C-repair	coNP-complete [2]
WA TGDs + EGDs	C-repair	coNP-complete [2]
DCs	CC-repair	coNP-complete [2]
full TGDs + EGDs	CC-repair	coNP-complete [2]
WA TGDs + EGDs	CC-repair	coNP-complete [2]
fixed FDs	U-repair	coNP-complete [14]
fixed CINDs	U-repair	coNP-complete [14]

Table 2: Complexity of repair checking

$(\mathcal{C}, T)$ is to decide, given a finite set $\Sigma \subseteq \mathcal{C}$ of dependencies defined over a relational schema $\mathcal{R}$, and two instances D and D_r of $\mathcal{R}$, whether D_r is a repair of D relative to Σ and $\mathsf{cost}_T(,)$?

The repair checking problem has been studied for various dependencies and repair models; some of the complexity bounds are presented in Table 2. Here a set of TGDs is said to be weakly acyclic (WA) if its dependency graph does not have a cycle going through a special edge that indicates an existentially quantified variable in Σ (see [2] for details).

Table 2 tells us that data repairing is rather expensive, especially for U-repair when attribute values are allowed to be updated: following [14], one can show that its data complexity is already intractable when only FDs or INDs are used.

2.2 Data Deduplication

Data deduplication is the problem of identifying tuples from one or more (possibly unreliable) relations that refer to the same real-world entity. It is also known as record matching, record linkage, entity resolution, instance identification, duplicate identification, merge-purge, database hardening, name matching, coreference resolution, identity uncertainty, and object identification. It is a longstanding issue that has been studied for decades [49], and is perhaps the most extensively studied data quality problem.

The need for data deduplication is evident in, *e.g.*, data quality management, data integration and fraud detection. It is particularly important to big data, which is often characterized by a large number of (heterogeneous) data sources. To make practical use of the data, it is often necessary to accurately identify tuples from different sources that refer to the same entity, so that we can fuse the data and enhance the information about the entity. This is nontrivial: data from various sources may be dirty, and moreover, even when the data sources are seemingly reliable, inconsistencies and conflicts often emerge when we integrate the data [14].

A variety of approaches have been proposed for data deduplication: probabilistic (*e.g.*, [49, 65, 95]), learning-based [27, 82], distance-based [60], and rule-based [3, 44, 61] (see [33, 62, 78] for surveys). In this paper we focus on rule-based collective and collaborative deduplication.

Data deduplication. To simplify the discussion, consider a single relation schema R. This does not lose generality since for any relational schema $\mathcal{R} = (R_1, \ldots, R_n)$, one can construct a single relation schema R and a linear bijective function $f()$ from instances of $\mathcal{R}$ to instances of R, without loss of information. Consider a set E of *entity types*, each specified by $e[X]$, where X is a set of attributes of R.

Given an instance D of R and a set E of entity types, *data deduplication* is to determine, for all tuples t, t' in D, and for each entity type $e[X]$, whether $t[X]$ and $t'[X]$ should be identified, *i.e.*, they refer to the same entity of type e. Following [13], we call $t[X]$ and $t'[X]$ *references* to e entities.

EXAMPLE 3. *On the employee relation of Figure 1, we may consider two entity types:* address *specified by* $(\mathsf{CC}, \mathsf{street}, \mathsf{city}, \mathsf{zip})$, *and* person *as the list of all attributes of employee. Given employee tuples t and t', deduplication is to decide whether $t[\mathsf{address}]$ and $t'[\mathsf{address}]$ refer to the same address, and whether t and t' are the same person.* □

As observed in [13], references to different entities may co-occur, and entities for co-occurring references should be determined jointly. For instance, papers and authors co-occur; identifying two authors helps identify their papers, and vice versa. This is referred to as *collective entity resolution* (deduplication) [13]. A graph-based method is proposed in [13] to propagate similarity among references, for collective deduplication. A datalog-like language is introduced in [5], with recursive rules for collective deduplication.

Matching rules. Rules were first studied in [3] for deduplication. Extending [3], a class of *matching dependencies* is defined in [44] in terms of similarity predicates and a matching operator $\rightleftharpoons$, based on a dynamic semantics [34].

EXAMPLE 4. *Matching dependencies on the employee relation of Figure 1 include the following:*

$$\psi_1 = \forall t, t' (t[\mathsf{CC}, \mathsf{AC}, \mathsf{landline}] = t'[\mathsf{CC}, \mathsf{AC}, \mathsf{landline}] \rightarrow t[\mathsf{address}] \rightleftharpoons t'[\mathsf{address}]),$$

$\psi_2 = \forall t, t'(t[\mathsf{LN}, \mathsf{address}] = t'[\mathsf{LN}, \mathsf{address}] \wedge t[\mathsf{FN}] \approx t'[\mathsf{FN}] \rightarrow t[\mathsf{person}] \rightleftharpoons t'[\mathsf{person}])$,
$\psi_3 = \forall t, t'(t[\mathsf{CC}, \mathsf{AC}, \mathsf{mobile}] = t'[\mathsf{CC}, \mathsf{AC}, \mathsf{mobile}] \rightarrow t[\mathsf{person}] \rightleftharpoons t'[\mathsf{person}])$,

Intuitively, (a) ψ_1 states that if t and t' have the same landline phone, then t[address] and t'[address] should refer to the same address and be equalized via updates; (b) ψ_2 says that if t and t' have the same address and last name, and if they have similar *first names, then they refer to the same person; and (c) ψ_3 states that if t and t' have the same mobile phone, then they should be identified as the same person. Here $\approx$ denotes a predicate for similarity of* FN*, such that, e.g., Bob $\approx$ Robert, since Bob is a nickname of Robert.*

These rules identify t_4 and t_5 in Figure 1 as follows. (a) By ψ_1, t_4[address] and t_5[address] should be identified although their values are radically different*; and (b) by (a) and ψ_2, t_4 and t_5 refer to the same person. Note that matching dependencies can be "recursively" applied: the outcome of (a) is used to deduce (b), for collective deduplication.* □

There exists a sound and complete axiom system for deducing matching dependencies from a set of known matching dependencies, based on their dynamic semantics [34]. The deduction process is in quadratic time. Moreover, "negative rules" such as "a male and a female cannot be the same person" can be expressed as matching dependencies without the need for introducing negation [45].

An operational semantics is developed for matching dependencies in [12] by means of a chase process with matching functions. It is shown that matching dependencies can also be used in data cleaning, together with related complexity bounds for consistent query answering [12]. Other types of rules have also been studied in, *e.g.*, [4, 16, 89].

Collaborative deduplication. Data repairing and deduplication are often taken as separate processes. To improve the accuracy of both processes, the two should be unified [45].

EXAMPLE 5. *We show how data repairing and deduplication interact to identify t_1–t_3 of Figure 1 as follows.*

(a) By CFD *φ_1 of Example 2, we have that t_2 and t_3 have the same address. By matching dependency ψ_2 of Example 4, we deduce that t_2 and t_3 refer to the same person. Moreover, we can enrich t_2 by t_2[landline, mobile] := t_3[landline, mobile].*

(b) By ψ_3 of Example 4, we deduce that t_1 and t_3 refer to the same person. Therefore, t_1–t_3 refer to the same Mary. □

The example shows that repairing helps deduplication and vice versa. This is also observed in [5]. Algorithms for unifying repairing and deduplication are given in [45]. In addition to data consistency, it has also been verified that data deduplication should also be combined with the analyses of data currency (timeliness) and data accuracy [42, 70].

Putting these together, we advocate *collaborative deduplication* that incorporates the analyses of data consistency (repairing), currency, accuracy and co-occurrences of attributes into the deduplication process, not limited to co-occurring references considered in collective deduplication [13].

2.3 Information Completeness

Information completeness concerns whether our database has complete information to answer our queries. Given a database D and a query Q, we want to know whether Q can be correctly answered by using only the data in D.

A database is typically assumed either closed or open.

- Under the Closed World Assumption (CWA), our database includes all the tuples representing real-world entities, but some *attribute values* may be *missing*.
- Under the Open World Assumption (OWA), our database may only be a proper subset of the set of tuples that represent real-world entities. That is, both tuples and values may be missing.

The CWA is often too strong in the real world [76]. Under the OWA, however, few queries can find correct answers.

To deal with missing values, representation systems are typically used (*e.g.*, c-tables, v-tables [59, 64]), based on certain query answers, which are recently revised in [71]. There has also been work on coping with missing tuples, by assuming that there exists a virtual database D_c with "complete information", and that part of D is known as a view of D_c [69, 77, 80]. Given such a database D, we want to determine whether a query posed on D_c can be answered by an equivalent query on D, via query answering using views.

Relative information completeness. We can possibly do better by making use of master data. An enterprise nowadays typically maintains *master data* (*a.k.a. reference data*), a single repository of high-quality data that provides various applications with a synchronized, consistent view of *the core business entities* of the enterprise [74].

Given a database D and master data D_m, we specify a set V of *containment constraints* [36]. Such a constraint

RCDP($\mathcal{L}_Q, \mathcal{L}_C$)	combined complexity [36]	data complexity [17]
(FO, CQ)	undecidable	undecidable
(CQ, CQ)	Π_2^p-complete	PTIME
(UCQ, UCQ)	Π_2^p-complete	PTIME

Table 3: Relative information completeness

is of the form $q(D) \subseteq p(D_m)$, where q is a query on D, and p is a simple projection on D_m. Intuitively, D_m is closed-world, and the part of D that is constrained by V is bounded by D_m, while the rest is open-world. We refer to a database D that satisfies V as a *partially closed* database *w.r.t.* (D_m, V). A database D_e is a *partially closed extension* of D if $D \subseteq D_e$ and D_e is partially closed *w.r.t.* (D_m, V) itself.

A partially closed database D is said to be *complete for a query Q relative to* (D_m, V) if for all partially closed extensions D_e of D *w.r.t.* (D_m, V), $Q(D_e) = Q(D)$. That is, there is no need for adding new tuples to D, since they either violate the containment constraints, or do not change the answer to Q in D. In other words, D already contains complete information necessary for answering Q [36].

EXAMPLE 6. *Recall that relation D_0 of Figure 1 may not have complete information to answer query Q_1 of Example 1. Now suppose that we have a master relation D_m of schema* (FN, LN, city)*, which maintains complete employee records in the UK, and a containment constraint ϕ: $\pi_{\mathsf{FN,LN,city}}\sigma_{CC=44}(D_0) \subseteq D_m$, i.e., the set of UK employees in D_0 is contained in D_m. Then if $Q_1(D_0)$ returns all employees in Edinburgh found in D_m, we can safely conclude that D_0 is complete for Q_1 relative to* $(D_m, \{\phi\})$. □

Several problems have been studied for relative information completeness [17, 36]. One of the problems, denoted by RCDP($\mathcal{L}_Q, \mathcal{L}_C$), is to determine, given a query Q, master data D_m, a set V of containment constraints, and a partially closed database D *w.r.t.* (D_m, V), whether D is complete for Q relatively to (D_m, V), where $\mathcal{L}_Q$ and $\mathcal{L}_C$ are query languages in which Q and q (in containment constraints) are expressed, respectively. Some complexity bounds of RCDP($\mathcal{L}_Q, \mathcal{L}_C$) are shown in Table 3, where CQ, UCQ and FO denote conjunctive queries (SPJ), unions of conjunctive queries (SPJU) and FO queries (the full relational algebra), respectively. The complexity bounds demonstrate the difficulty of reasoning about information completeness. Relative information completeness has also been studied in the setting where both values and tuples may be missing, by extending representation systems for missing values [35].

Containment constraints are also able to express dependencies used in the analysis of data consistency, such as CFDs and CINDs [36]. Hence we can study data consistency and information completeness in a uniform framework.

2.4 Data Currency

Data currency (*timeliness*) aims to identify the current values of entities represented by tuples in a (possibly stale) database, and to answer queries with the current values.

There has been work on how to define current tuples by means of timestamps in temporal databases (see, *e.g.*, [24, 83] for surveys). In practice, however, timestamps are often unavailable or imprecise [96]. The question is how to determine data currency in the absence of reliable timestamps.

Modeling data currency. We present a model proposed in [43]. Consider a database D that possibly contains stale data. For each tuple $t \in D$, $t[\mathsf{eid}]$ denotes the id of the entity that t represents, obtained by data deduplication (see Section 2.2).

(1) The model assumes *a currency order* $\prec_A$ for each attribute A of each relation schema R, such that for tuples t_1 and t_2 of schema R in D, if $t_1[\mathsf{eid}] = t_2[\mathsf{eid}]$, *i.e.*, when t_1 and t_2 represent the same entity, then $t_1 \prec_A t_2$ indicates that t_2 is more up-to-date than t_1 in the A attribute value. This is to model partially available currency information in D.

(2) The model uses *currency constraints* to specify currency relationships derived from the semantics of the data, expressed as denial constraints equipped with constants.

EXAMPLE 7. *Extending relation D_0 of Figure 1 with attribute* eid*, currency constraints on D_0 include:*

$$\forall s, t\big((s[\mathsf{eid}] = t[\mathsf{eid}] \wedge s[\mathsf{status}] = \text{“}\textit{married}\text{”} \wedge t[\mathsf{status}] = \text{“}\textit{single}\text{”}) \rightarrow t \prec_{\mathsf{status}} s\big),$$
$$\forall s, t\big((s[\mathsf{eid}] = t[\mathsf{eid}] \wedge t \prec_{\mathsf{status}} s \rightarrow t \prec_{\mathsf{LN}} s\big).$$

These constraints are derived from the semantics of the data: (a) marital changes from “single” to “married”, but not the other way around; and (b) LN *and* status *are correlated: if t has more current status than s, it also has more current* LN.

Based on these, query Q_2 of Example 1 can be answered with the most current LN *value of Mary, namely, Luth.* □

Based on currency orders and constraints, we can define (3) *consistent completions* D^c of D, which extend $\prec_A$ in D to a total order on all tuples pertaining to the same

CCQA($\mathcal{L}_Q$)	combined complexity [43]	data complexity [43]
FO	PSPACE-complete	coNP-complete
CQ, UCQ	Π_2^p-complete	coNP-complete

Table 4: Certain current answers

entity, such that D^c satisfies the currency constraints; and

(4) from D^c, we can extract the *current tuple* for each entity eid, composed of the entity's most current A value for each attribute A based on $\prec_A$. This yields the *current instance* of D^c consisting of only the current tuples of the entities in D, from which currency orders can be removed.

(5) We compute *certain current answers* to a query Q in D, *i.e.,* answers to Q in *all* consistent completions D^c of D.

Several problems associated with data currency are studied in [43]. One of the problems, denoted by CCQA($\mathcal{L}_Q$), is to decide, given a database D with partial currency orders $\prec_A$ and currency constraints, a query $Q \in \mathcal{L}_Q$ and a tuple t, whether t is a certain current answer to Q in D. Some of the complexity results for CCQA($\mathcal{L}_Q$) are shown in Table 4.

2.5 Data Accuracy

Data accuracy refers to the closeness of values in a database to the true values of the entities that the data in the database represents, when the true values are not known.

While it has long been recognized that data accuracy is critical to data quality [7], the topic has not been well studied. Prior work typically studies the reliability of data sources, *e.g.,* dependencies [30] and lineage information [90] of data sources to detect copy relationships and identify reliable sources, vote counting and probabilistic analysis based on the trustworthiness of data sources [51, 97].

Complementary to the reliability analysis of sources, relative accuracy is studied in [18]. Given tuples t_1 and t_2 that pertain to the same entity, it is to infer whether $t_1[A]$ is more accurate than $t_2[A]$ for attributes A of the tuples. The inference is conducted by a chase process, by combining the analyses of data consistency, currency and correlated attributes.

3. DATA CLEANING TECHNIQUES

As Gartner [54] put it, the data quality tool market is "among the fastest-growing in the enterprise software sector". It reached $1.13 billion in software revenue in 2013, about 13.2% growth, and will reach $2 billion by 2017, 16% growth. While data quality tools have mostly been dealing with customer, citizen and patient data, they are rapidly expanding into financial and quantitative data domains.

What does the industry need from data quality tools? Such tools are expected to automate key elements, including: (1) data profiling to discover data quality rules, in particular "dependency analysis (cross-table and cross-dataset analysis)"; (2) cleaning, "the modification of data values to meet domain restrictions, integrity constraints or other business rules"; and (3) matching, "the identifying, linking and merging of related entries within or across sets of data", and in particular, "matching rules or algorithms" [54].

In this section we briefly survey techniques for profiling (discovery of data quality rules), cleaning (error detection and data repairing) and matching (data deduplication).

3.1 Discovering Data Quality Rules

To clean data with data quality rules, the first question we have to answer is how we can get the rules. It is unrealistic to rely on domain experts to design data quality rules via an expensive and long manual process, or count on business rules that have been accumulated. This highlights the need for automatically *discovering* and *validating* data quality rules.

Rule discovery. For a class $\mathcal{C}$ of dependencies that are used as data quality rules, the *discovery problem* for $\mathcal{C}$ is stated as follows. Given a database instance D, it is to find a *minimal cover*, a non-redundant set of dependencies that is logically equivalent to the set of all dependencies in $\mathcal{C}$ that hold on D.

A number of discovery algorithms are developed for, *e.g.,*

- FDs, *e.g.,* [63, 93], and INDs (see [72] for a survey);
- CFDs, *e.g.,* [21, 39, 56, 58], and CINDs [56];
- denial constraints DCs [25]; and for
- matching dependencies [84].

Discovery algorithms are often based on the levelwise approach proposed by [63], *e.g.,* [21, 39], depth-first search of [93], *e.g.,* [25, 39], and association rule mining [39, 56].

Rule validation. Data quality rules are discovered from possibly dirty data, and are likely "dirty" themselves. Hence given a set Σ of discovered rules, we need to identify what rules in Σ make sense, by checking their satisfiability. In addition, we want to remove redundant rules from Σ, by making use of implication analysis (see Section 2.1).

It is nontrivial to identify sensible rules from Σ. Recall that the satisfiability problem is NP-complete for CFDs, and is nontrivial for DCs. Nevertheless, approximation algorithms can be developed. For CFDs, such algorithms have been studied [38], which extract a set Σ' of satisfiable dependencies from Σ, and guarantee that Σ' is "close" to a maximum satisfiable subset of Σ, within a constant bound.

3.2 Error Detection

After data quality rules are discovered and validated, the next question concerns how to effectively catch errors in a database by using these rules. Given a database D and a set Σ of dependencies as data quality rules, *error detection* (*a.k.a. error localization*) is to find all tuples in D that violate at least one dependency in Σ. Error detection is a routine operation of data quality tools. To clean data we have to detect errors first. Many users simply want errors in their data to be detected, without asking for repairing the data.

Error detection methods depend on (a) what dependencies are used as data quality rules, and (b) whether the data is stored in a local database or distributed across different sites.

Centralized databases. When D resides in a centralized database and when Σ is a set of CFDs, *two SQL queries* Q^c and Q^v can be *automatically* generated such that $Q^c(D)$ and $Q^v(D)$ return all and only those tuples in D that violate Σ [38]. Better still, Q^c and Q^v are independent of the number and size of CFDs in Σ. That is, we can detect errors by leveraging existing facility of commercial relational DBMS.

EXAMPLE 8. *To detect violations of* $\varphi_2 = ((\mathsf{CC}, \mathsf{AC} \to \mathsf{city}),\ T_{P2})$ *of Example 2, we use the following* Q^c *and* Q^v*:*

Q^C
```
SELECT * FROM R t, T_P2 t_p
WHERE t[CC, AC] ≍ t_p[CC, AC] AND t[city] ≭ t_p[city]
```

Q^V
```
SELECT DISTNCT CC, AC FROM R t, T_P2 t_p
WHERE  t[CC, AC] ≍ t_p[CC, AC] AND t_p[city] = '_'
GROUP BY CC, AC HAVING COUNT(DISTNCT city) > 1
```

where $t[\mathsf{CC}, \mathsf{AC}] \asymp t_p[\mathsf{CC}, \mathsf{AC}]$ *denotes (*$t[\mathsf{CC}] = t_p[\mathsf{CC}]$ *OR* $t_p[\mathsf{CC}] =$ '_'*) AND (*$t[\mathsf{AC}] = t_p[\mathsf{AC}]$ *OR* $t_p[\mathsf{AC}] =$ '_'*); and* R *denotes the schema of employee datasets. Intuitively,* Q^C *catches single-tuple violations of* φ_2*, i.e., those that violate a pattern in* T_{P2}*, and* Q^V *identifies violations of the* FD *embedded in* φ_2*. Note that* Q^C *and* Q^V *simply treat pattern tableau* T_{P2} *as an "input" relation, regardless of its size. In other words,* Q^C *and* Q^V *are determined only by the* FD *embedded in* φ_2*, no matter how large the tableau* T_{P2} *is.*

When Σ *consists of multiple* CFDs*, we can "merge" these* CFDs *into an equivalent one, by making use of a new wildcard [38]. Thus two SQL queries as above suffice for* Σ*.* □

The SQL-based method also works for CINDs [20].

Distributed data. In practice a database is often fragmented and distributed across different sites. In this setting, error detection necessarily requires data shipment from one site to another. For both vertically or horizontally partitioned data, it is NP-complete to decide whether error detection can be carried out by shipping a bounded amount of data, and the SQL-based method no longer works [40]. Nevertheless, distributed algorithms are in place to detect CFD violations in distributed data, with performance guarantees [40, 47].

3.3 Data Repairing

After errors are detected, we want to fix the errors. Given a database D and a set Σ of dependencies as data quality rules, *data repairing* (*a.k.a. data imputation*) is to find a repair D_r of D with minimum $\mathsf{cost}(D, D_r)$. We focus on the U-repair model based on attribute-value modifications (see Section 2.1), as it is widely used in the real world [54].

Heuristic fixes. Data repairing is cost-prohibitive: its data complexity is coNP-complete for fixed FDs or INDs [14]. In light of this, repairing algorithms are mostly heuristic, by enforcing dependencies in Σ one by one. This is nontrivial.

EXAMPLE 9. *Consider two relation schemas* $R_1(A, B)$ *and* $R_2(B, C)$*, an* FD *on* R_1*:* $A \to B$*, and an* IND $R_2[B] \subseteq R_1[B]$*. Consider instances* $D_1 = \{(1,2), (1,3)\}$ *of* R_1 *and* $D_2 = \{(2,1), (3,4)\}$*, where* D_1 *does not satisfy the* FD*. To repair* (D_1, D_2)*, a heuristic may enforce the* FD *first, to "equalize"* 2 *and* 3*; it then needs to enforce the* IND*, by ensuring that* D_1 *includes* $\{2, 3\}$ *as its* B*-attribute values. This yields a repairing process that does not terminate.* □

Taking both FDs and INDs as data quality rules, a heuristic method is proposed in [14] based on equivalence classes, which group together attribute values of D that must take the same value. The idea is to separate the decision of which values should be equal from the decision of what values should be assigned to the equivalence classes. Based on the $\mathsf{cost}(,)$ function given in Section 2.1, it guarantees to find a repair. The method has been extended to repair data based on CFDs [28], EGDs and TGDs [55] with a partial order on equivalence classes to specify preferred updates, and DCs [26] by generalizing equivalence classes to conflict hypergraphs. An approximation algorithm for repairing data based on FDs was developed in [66].

A semi-automated method is introduced in [94] for data repairing based on CFDs. In contrast to [14], it interacts with users to solicit credible updates and improve the accuracy. Another repairing method is studied in [45], which picks reliable fixes based on an analysis of the relative certainty of the data, measured by entropy. There have also been attempts to unify data repairing and deduplication [45] based on CFDs, matching dependencies and master data.

Certain Fixes. A major problem with heuristic repairing methods is that they do not guarantee to find correct fixes; worse still, they may introduce new errors when attempting to fix existing errors. As an example, to fix tuple t_1 of Figure 1 that violates CFD φ_2 of Example 2, a heuristic method may very likely change $t_1[\mathsf{city}]$ from London to Edinburgh. While the change makes t_1 a "repair", the chances are that for the entity represented by t_1, AC is 020 and city is London. That is, the heuristic update does not correct the error in $t_1[\mathsf{AC}]$, and worse yet, it changes $t_1[\mathsf{city}]$ to a wrong value. Hence, while the heuristic methods may suffice for statistical analysis, *e.g.*, census data, they are often too risky to be used in repairing critical data such as medical records.

This highlights the need for studying certain fixes for critical data, *i.e.*, fixes that are guaranteed to be correct [46]. To identify certain fixes, we make use of (a) master data (Section 2.3), (b) editing rules instead of data dependencies, and (c) a chase process for inferring "certain regions" based on user confirmation, master data and editing rules, where certain regions are attribute values that are validated.

Editing rules are dynamic constraints that tell us which attributes should be changed and to what values they should be changed. In contrast, dependencies have a static semantics; they are capable of detecting the presence of errors in the data, but they do not tell us how to fix the errors.

EXAMPLE 10. *Assume master data D_m with schema $R_m(\mathsf{postal}, \mathsf{C}, \mathsf{A})$ for postal code, city and area code in the UK. An editing rule for D_0 of Fig. 1 is as follows:*

$$\sigma\text{:}\ (\mathsf{postal}, \mathsf{zip}) \rightarrow ((\mathsf{C}, \mathsf{city}), (\mathsf{A}, \mathsf{AC})),$$

specified with pairs of attributes from D_m and D_0. It states that for an input tuple t, if $t[\mathsf{zip}]$ is validated and there exists a master tuple $s \in D_m$ such that $t[\mathsf{zip}] = s[\mathsf{postal}]$, then update $t[\mathsf{city}, \mathsf{AC}] := s[\mathsf{C}, \mathsf{A}]$ is guaranteed a certain fix, and $t[\mathsf{AC}, \mathsf{city}]$ becomes a certain region (validated). Suppose that there is s = (W1B 1JL, London, 020) in D_m, and that $t_1[\mathsf{zip}]$ of Figure 1 is validated. Then $t_1[\mathsf{AC}]$ should be changed to 020; here $t_1[\mathsf{city}]$ remains unchanged. □

A framework is developed in [46] for inferring certain fixes for input tuples. Although it may not be able to fix all the errors in the data based on available information, it guarantees that each update fixes at least one error, and that no new errors are introduced in the entire repairing process. The process may consult users to validate a minimum number of attributes in the input tuples. Static analyses of editing rules and certain regions can also be found in [46].

Editing rules are generalized in [29] by allowing generic functions to encompass editing rules [46], CFDs and matching dependencies. However, it remains to be justified whether such generic rules can be validated themselves and whether the fixes generated are sensible at all.

Beyond data repairing. Data repairing typically assumes that data quality rules have been validated. Indeed, in practice we use data quality rules to clean data only after the rules are confirmed correct themselves. A more general setting is studied in [22], when both data and data quality rules are possibly dirty and need to be repaired.

There has also been work on (a) causality of errors [75] and its connection with data repairs [81], and (b) propagation of errors and dependencies in data transformations [19, 48].

3.4 Data Deduplication

A number of systems have been developed for data deduplication, *e.g.*, BigMatch [95], Tailor [32], Swoosh [10] AJAX [50], CrowdER [88] and Corleone [57], as stand-alone tools, embedded packages in ETL systems, or crowd-sourced systems. Criteria for developing such systems include (a) accuracy, to reduce *false matches* (false positives) and *false non-matches* (false negatives); and (b) scalability with big data. To improve accuracy, we advocate collaborative deduplication (Section 2.2), including but not limited to collective deduplication [13]. For scalability, parallel matching methods need to be developed and combined with traditional blocking and windowing techniques (see [33]). We refer the interested reader to [62, 78] for detailed surveys.

4. CHALLENGES INTRODUCED BY BIG DATA

The study of data quality has raised as many questions as it has answered. In particular, a full treatment is required for each of data accuracy, currency and information completeness, as well as their interaction with data consistency and deduplication. Moreover, big data introduces a number of challenges, and the study of big

data quality is in its infancy.

Volume. Cleaning big data is cost-prohibitive: discovering data quality rules, error detection, data repairing and data deduplication are all expensive; *e.g.,* the data complexity of data repairing is coNP-complete for FDs and INDs [14]. To see what it means in the context of big data, observe that a linear scan of a dataset D of PB size (10^{15} bytes) takes days using a solid state drive with a read speed of 6GB/s, and it takes years if D is of EB size (10^{18} bytes) [41].

To cope with the volume of big data, we advocate the following approaches, taking data repairing as an example.

Parallel scalable algorithms. We approach big data repairing by developing parallel algorithms. This is often necessary since in the real world, big data is often distributed.

It is *not* always the case that the more processors are used, the faster we get. To characterize the effectiveness of parallelization, we formalize parallel scalability following [68].

Consider a dataset D and a set Σ of data quality rules. We denote by $t(|D|, |\Sigma|)$ the worst-case running time of a *sequential algorithm* for repairing D with Σ; and by $T(|D|, |\Sigma|, n)$ the time taken by a parallel algorithm for the task *by using* n *processors*, taking n as a parameter. Here we assume $n \ll |D|$, *i.e.,* the number of processors does not exceed the size of the data, as commonly found in practice.

We say that the algorithm is *parallel scalable* if

$$T(|D|, |\Sigma|, n) = O(t(|D|, |\Sigma|)/n) + (n|\Sigma|)^{O(1)}.$$

That is, the parallel algorithm achieves a polynomial reduction in sequential running time, plus a "bookkeeping" cost $O((n|\Sigma|)^l)$ for a constant l that is *independent of* $|D|$.

Obviously, if the algorithm is parallel scalable, then for a given D, it *guarantees* that the more processors are used, the less time it takes to repair D. It allows us to repair big data by adding processors when needed. If an algorithm is not parallel scalable, it may not be able to efficiently repair D when D grows big *no matter how many* processors are used.

Entity instances. We propose to deal with entity instances instead of processing the big dataset D directly. An *entity instance* I_e is a set of tuples in D that pertain to the same entity e. It is *substantially smaller* than D, and typically retains a manageable size when D grows big. This suggests the following approach to repairing big data: (1) cluster D into entity instances I_e, by using a parallel data deduplication algorithm; (2) for each entity e, deduce "the true values" of e from I_e, by processing all entities in parallel; and (3) resolve inconsistencies across different entities, again in parallel.

We find that this approach allows us to effectively and efficiently deduce accurate values for each entity, by reasoning about data consistency, data deduplication with master data, data accuracy and data currency together [18, 42].

Bounded incremental repairing. We advocate *incremental data repairing*. Given a big dataset D, a set Σ of data quality rules, a repair D_r of D with Σ, and updates ΔD to D, it is to find *changes* ΔD_r to the repair D_r such that $D_r \oplus \Delta D_r$ is a repair of $D \oplus \Delta D$ with Σ, where $D \oplus \Delta D$ denotes the updated dataset of D with ΔD; similarly for $D_r \oplus \Delta D_r$.

Intuitively, small changes ΔD to D often incur a small number of new violations to the rules in Σ; hence, changes ΔD_r to the repair D_r are also small, and it is more efficient to find ΔD_r than to compute a new repair starting from scratch. In practice, data is frequently updated, but the changes ΔD are typically small. We can minimize unnecessary recomputation of D_r by incremental data repairing.

The benefit is more evident if there exists a bounded incremental repairing algorithm. As argued in [79], incremental algorithms should be analyzed in terms of |CHANGED| = $|\Delta D|$ + $|\Delta D_r|$, indicating the updating costs that are *inherent to* the incremental problem itself. An incremental algorithm is said to be *bounded* if its cost can be expressed as a function of |CHANGED| and $|\Sigma|$, *i.e.,* it depends only on |CHANGED| and $|\Sigma|$, *independent of* the size of big D.

This suggests the following approach to repairing and maintaining a big dataset D. (1) We compute repair D_r of D *once*, in parallel by using a number of processors. (2) In response to updates ΔD to D, we *incrementally* compute ΔD_r, by reducing the problem of repairing big D to an incremental problem on "small data" of size |CHANGED|. The incremental step may not need a lot of resources.

Besides the scalability of repairing algorithms with big data, we need to ensure the accuracy of repairs. To this end, we promote the following approach.

Knowledge bases as master data. Master data is extremely helpful in identifying certain fixes [46], data repairing [45] and in deducing the true values of entities [18, 42]. A number of high-quality knowledge bases are already developed these days, and can be employed as master data. We believe that repairing algorithms should be developed by taking the knowledge bases as master data, to improve the accuracy.

Velocity. Big datasets are "dynamic": they change fre-

quently. This further highlights the need for developing bounded incremental algorithms for data cleaning. When CFDs are used as data quality rules, incremental algorithms are in place for error detection in centralized databases [38] and distributed data [47], and for data repairing [28]. Nonetheless, parallel incremental algorithms need to be developed for error detection, data repairing and deduplication.

Variety. Big data is also characterized by its heterogeneity. Unfortunately, very little is known about how to model and improve the quality of data beyond relations. In particular, graphs are a major source of big data, *e.g.*, social graphs, knowledge bases, Web sites, and transportation networks. However, integrity constraints are not yet well studied for graphs to determine the consistency of the data. Even keys, a primary form of data dependencies, are not yet defined for graphs. Given a graph G, we need keys that help us uniquely identify entities represented by vertices in G.

Keys for graphs are, however, a departure from their counterparts for relations, since such keys have to be specified in terms of both attribute values of vertices and the topological structures of neighborhoods, perhaps in terms of graph pattern matching by means of subgraph isomorphism.

Acknowledgments. The author thanks Floris Geerts for his thorough reading of the first draft and for helpful comments. The author is supported in part by NSFC 61133002, 973 Program 2012CB316200, ERC 652976, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, EPSRC EP/J015377/1 and EP/M025268/1, NSF III 1302212, and a Google Faculty Research Award.

5. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.
[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.
[4] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, pages 40–49, 2008.
[5] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963, 2009.
[6] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.
[7] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.
[8] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *JCSS*, 59(1):94–115, 1999.
[9] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *JACM*, 31(4):718–741, 1984.
[10] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009.
[11] L. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.
[12] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *TCS*, 52(3):441–482, 2013.
[13] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 1(1), 2007.
[14] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.
[15] L. Bravo, W. Fan, and S. Ma. Extending inclusion dependencies with conditions. In *VLDB*, 2007.
[16] D. Burdick, R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. A declarative framework for linking entities. In *ICDT*, 2015.
[17] Y. Cao, T. Deng, W. Fan, and F. Geerts. On the data complexity of relative information completeness. *Inf. Syst.*, 45:18–34, 2014.
[18] Y. Cao, W. Fan, and W. Yu. Determining the relative accuracy of attributes. In *SIGMOD*, pages 565–576, 2013.
[19] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, pages 445–456, 2014.
[20] W. Chen, W. Fan, and S. Ma. Analyses and validation of conditional dependencies with built-in predicates. In *DEXA*, 2009.
[21] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.
[22] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *ICDE*, pages 446–457, 2011.
[23] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, 2005.
[24] J. Chomicki and D. Toman. Time in database systems. In M. Fisher, D. Gabbay, and L. Víla, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 429–467. Elsevier, 2005.
[25] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
[26] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[27] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *KDD*, 2002.
[28] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.
[29] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, 2013.
[30] X. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. In *PVLDB*, 2009.
[31] W. W. Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. Technical report, The Data Warehousing Institute, 2002.
[32] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. TAILOR: A record linkage tool box. In *ICDE*, 2002.
[33] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
[34] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *VLDB J.*, 20(4):495–520, 2011.
[35] W. Fan and F. Geerts. Capturing missing tuples and missing values. In *PODS*, pages 169–178, 2010.
[36] W. Fan and F. Geerts. Relative information completeness. *ACM Trans. on Database Systems*, 35(4), 2010.
[37] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.
[38] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies.

TODS, 33(1), 2008.
[39] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.
[40] W. Fan, F. Geerts, S. Ma, and H. Müller. Detecting inconsistencies in distributed data. In *ICDE*, pages 64–75, 2010.
[41] W. Fan, F. Geerts, and F. Neven. Making queries tractable on big data with preprocessing. *PVLDB*, 6(8):577–588, 2013.
[42] W. Fan, F. Geerts, N. Tang, and W. Yu. Conflict resolution with data currency and consistency. *J. Data and Information Quality*, 5(1-2):6, 2014.
[43] W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. *TODS*, 37(4), 2012.
[44] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. In *VLDB*, 2009.
[45] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.
[46] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.
[47] W. Fan, J. Li, N. Tang, and W. Yu. Incremental detection of inconsistencies in distributed data. *TKDE*, 2014.
[48] W. Fan, S. Ma, Y. Hu, J. Liu, and Y. Wu. Propagating functional dependencies with conditions. *PVLDB*, 1(1):391–407, 2008.
[49] I. Fellegi and A. B. Sunter. A theory for record linkage. *J. American Statistical Association*, 64(328):1183–1210, 1969.
[50] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: An extensible data cleaning tool. In *SIGMOD*, page 590, 2000.
[51] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *WSDM*, 2010.
[52] V. Ganti and A. D. Sarma. *Data Cleaning: A Practical Perspective*. Morgan & Claypool Publishers, 2013.
[53] Gartner. 'Dirty data' is a business problem, not an IT problem, 2007. *http://www.gartner.com/newsroom/id/501733.*
[54] Gartner. Magic quardrant for data quality tools, 2014.
[55] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 2013.
[56] B. Goethals, W. L. Page, and H. Mannila. Mining association rules of simple conjunctive queries. In *SDM*, 2008.
[57] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.
[58] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.
[59] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.
[60] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *VLDB*, 2004.
[61] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.
[62] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, 2009.
[63] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *COMP. J.*, 42(2):100–111, 1999.
[64] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.
[65] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa Florida. *J. American Statistical Association*, 89:414–420, 1989.
[66] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, 2009.
[67] P. G. Kolaitis and E. Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.
[68] C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *TCS*, 71(1):95–132, 1990.
[69] A. Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB*, pages 402–412, 1996.
[70] P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.
[71] L. Libkin. Certain answers as objects and knowledge. In *KR*, 2014.
[72] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data - a review. *TKDE*, 24(2):251–264, 2012.
[73] A. Lopatenko and L. E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pages 179–193, 2007.
[74] D. Loshin. *Master Data Management*. Knowledge Integrity, Inc., 2009.
[75] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, pages 505–516, 2011.
[76] D. W. Miller Jr., J. D. Yeast, and R. L. Evans. Missing prenatal records at a birth center: A communication problem quantified. In *AMIA Annu Symp Proc.*, pages 535–539, 2005.
[77] A. Motro. Integrity = validity + completeness. *ACM Trans. on Database Systems*, 14(4):480–502, 1989.
[78] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers, 2010.
[79] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2):213–224, 1996.
[80] S. Razniewski and W. Nutt. Completeness of queries over incomplete databases. *PVLDB*, pages 749–760, 2011.
[81] B. Salimi and L. E. Bertossi. From causes for database queries to repairs and model-based diagnosis and back. In *ICDT*, 2015.
[82] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.
[83] R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.
[84] S. Song and L. Chen. Efficient discovery of similarity constraints for matching dependencies. *TKDE*, 87:146–166, 2013.
[85] S. Staworko. *Declarative inconsistency handling in relational and semi-structured databases*. PhD thesis, the State University of New York at Buffalo, 2007.
[86] S. Staworko, J. Chomicki, and J. Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.
[87] B. ten Cate, G. Fontaine, and P. G. Kolaitis. On the data complexity of consistent query answering. In *ICDT*, pages 22–33, 2012.
[88] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing entity resolution. *PVLDB*, 2012.
[89] S. Whang, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with negative rules. *VLDB J.*, 18(6):1261–1277, 2009.
[90] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.
[91] J. Wijsen. Database repairing using updates. *TODS*, 30(3), 2005.
[92] Wikibon. A comprehensive list of big data statistics, 2012. *http://wikibon.org/blog/big-data-statistics/.*
[93] C. M. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *DaWak*, 2001.
[94] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, pages 279–289, 2011.
[95] W. Yancey. BigMatch: A program for extracting probable matches from a large file. Technical Report Computing 2007/01, U.S. Census Bureau, 2007.
[96] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *PVLDB*, pages 244–255, 2010.
[97] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 2012.